TP2 : Déduction par coupure

Objectifs du TP :

- Découvrir un autre système de déduction dans le cadre du calcul propositionnel.
- Ecrire un algorithme qui décide syntaxiquement d'un résultat sémantique.
- Réviser la manipulation de listes en Ocaml et penser récursif !

Dans cet énoncé, on se limite à manipuler un sous ensemble des formules du calcul propositionnel formé d'éléments qu'on appelle des *clauses*. Une clause est simplement une disjonction de littéraux. Les variables intervenant dans une clause C sans être précédée du connecteur $\neg$ sont appelées les variables positives de C, et les autres les variables négatives de C.

1. Rappeler pourquoi toute formule du calcul propositionnel est sémantiquement équivalente à une conjonction de clauses.
2. Si C est une clause dont les variables positives sont $a_1, ..., a_n$ et les variables négatives sont $b_1, ..., b_m$, montrer que C est équivalente à l'implication $(b_1 \wedge ... \wedge b_m) \Rightarrow (a_1 \vee ... \vee a_n)$.

Remarquez que le résultat précédent reste vrai, même s'il n'y a a aucune variable positive ou aucune variable négative si on considère qu'un $\wedge$ sur un ensemble vide vaut le neutre pour $\wedge$, à savoir $\top$ et que le $\vee$ sur un ensemble vide vaut le neutre pour $\vee$, à savoir $\bot$. Une clause ne contenant ni variable positive, ni variable négative est en particulier sémantiquement équivalente à $\top \Rightarrow \bot$ c'est à dire à $\bot$ et on appelera une telle clause la *clause vide*.

On ne considère dans la suite que des *clauses simplifiées*, c'est-à-dire des clauses sans littéral en doublon. Dès qu'une clause C fera intervenir un littéral en double, on commencera par supprimer les occurrences inutiles de ce doublon et on appelle cette opération la *simplification* de C. **Dans la suite, une clause est par défaut une clause simplifiée.**

La déduction par coupure est un système de déduction dont la seule règle est la règle dite de coupure. Pour toutes clauses C_1 et C_2, on dit que la clause C se déduit par coupure de C_1 et C_2 s'il existe une variable v positive dans C_1 et négative dans C_2 (resp. négative dans C_1 et positive dans C_2) et telle que C est la simplification de la disjonction de tous les littéraux de C_1 sauf v (resp. $\neg v$) et de tous les littéraux de C_2 sauf $\neg v$ (resp. v). On dit qu'on a coupé sur la variable v et on note :

$$\frac{C_1 \qquad C_2}{C}\ \text{cut}$$

Si S est un ensemble de clauses (simplifiées) initial, l'ensemble des clauses C prouvables par coupure à partir de S est défini inductivement par :

- Si $C \in S$, C est prouvable à partir de S.
- Si C_1 et C_2 sont prouvables par coupure à partir de S et que C se déduit par coupure à partir de C_1 et C_2 alors C est prouvable par coupure à partir de S.

Si C est prouvable par coupure à partir de S, on note $S \vdash C$.

Exemple : $\{\neg a \vee b \vee \neg c, c \vee b, a \vee \neg d\} \vdash b \vee \neg d$. En effet, on a :

$$\dfrac{\dfrac{\neg a \vee b \vee \neg c \qquad a \vee \neg d}{b \vee \neg c \vee \neg d}\ \text{cut} \qquad c \vee b}{b \vee \neg d}\ \text{cut}$$

Vu la question 2, cela ne devrait pas choquer votre intuition sémantique. En effet, $\neg a \vee b \vee \neg c \equiv a \Rightarrow (b \vee \neg c)$ et $a \vee \neg d \equiv d \Rightarrow a$. Il n'est donc pas anormal de pouvoir déduire $d \Rightarrow (b \vee \neg c) \equiv b \vee \neg d \vee \neg c$ lors de la première application de la règle de coupure dans l'arbre précédent.

Lorsque la clause vide est prouvable par coupure à partir de S, on dit que S admet une *réfutation par coupure*. On peut montrer que la réfutation par coupure est correcte et complète (cf partie 3) : autrement dit, $S \vdash \bot$ si et seulement S est non satisfiable.

Partie 1 *Implémentation de la réfutation par coupure*

L'objectif de cette partie est d'implémenter un algorithme derive_clause_vide dont la spécification est :

Entrée : Un ensemble de clauses (simplifiées) S.
Sortie : Oui si S admet une réfutation par coupure, non sinon.

D'après ce qui précède, un tel algorithme permet de décider si un ensemble de clauses est non satisfiable.

On propose de représenter une clause en Ocaml comme suit : un clause est une liste d'entiers relatifs non nuls. Un entier positif représente une variable positive et un entier négatif une variable négative. On demande de plus à une clause d'être une liste :

- Triée dans l'ordre croissant. Ceci sera nécessaire pour pouvoir tester l'égalité de deux clauses.
- Sans doublons, de sorte à ne considérer que des clauses simplifiées.

Par exemple, la clause $v_1 \vee v_2 \vee \neg v_4$ sera représentée par la liste [-4;1;2] et la clause vide par [].

```
type clause = int list
```

L'idée pour construire l'algorithme derive_clause_vide est le suivant : si S est un ensemble de clauses, on construit toutes les clauses prouvables par coupure à partir de S puis on vérifie si la clause vide fait partie de l'ensemble ainsi construit. Plus précisément :

- On maintient à jour un ensemble C_t de clauses à traiter et C_p de clauses prouvables. Initialement, C_t est l'ensemble de clauses initial et C_p est vide.
- Tant que C_t n'est pas vide, on en prend une clause, c. Pour tout élément $c' \in C_p$, on construit toutes les clauses déductibles par coupure de c et c'. Toutes les clauses ainsi construites qui ne sont ni déjà dans C_p, ni déjà dans C_t sont rajoutées à C_t. Une fois qu'on a généré toutes les clauses possibles à partir de c et des clauses de C_p, on ajoute c à C_p.
- Lorsque C_t est vide, on vérifie si la clause vide est dans C_p.

3. Ecrire une fonction coupure de signature clause -> clause -> int -> clause. Elle prend en entrée deux clauses c_1, c_2 et un entier v. On supposera que v apparaît dans c_1 et que son opposé apparaît dans c_2. Elle renvoie une liste triée sans doublon correspondant à la clause obtenue par coupure à partir de c_1 et c_2 sur la variable v.
4. Ecrire une fonction variables_a_couper de signature clause -> clause -> int list telle que variables_a_couper c1 c2 renvoie la liste des variables v de c1 telles qu'on puisse appliquer la règle de coupure à c1 et c2 en coupant sur v.
5. Déduire des questions précédentes une fonction nouvelles_clauses de signature clause -> clause -> clause list telle que nouvelles_clauses c1 c2 renvoie la liste des clauses déductibles par coupure à partir de c1 et c2.

6. Ecrire une fonction exists_clause_vide de signature clause list -> bool indiquant si la clause vide fait partie de la liste de clauses en entrée.

7. Déduire de tout ce qui précède une fonction derive_clause_vide de signature clause list -> bool telle que derive_clause_vide lc renvoie true si la clause vide est prouvable par coupure à partir de l'ensemble de clauses lc et false sinon.

   On pourra s'aider de fonctions auxilliares et des fonctions du module List.

La fonction précédente est excessivement naïve. Une amélioration immédiate serait d'ailleurs de cesser de construire des clauses prouvables dès que la clause vide a été obtenue. La fin de cette partie est un bonus dans lequel on explore deux stratégies visant à limiter la complexité en pratique de derive_clause_vide.

8. Jusqu'à présent, les clauses $c \in C_t$ avec lesquelles on tente successivement de créer de nouvelles clauses prouvables sont choisies au hasard. Comment pourrait-on choisir c pour espérer obtenir la clause vide rapidement ? Proposer une heuristique de choix de c et l'implémenter.

9. Une clause simplifiée est dite tautologique si elle fait intervenir une variable à la fois positivement et négativement.

   a) Montrer que, si $S \vdash C$ alors la preuve par coupure de C à partir de S peut se faire sans utiliser de clause tautologique. On en déduit qu'ajouter les clauses tautologiques à C_t ou C_p est inutile à la correction de la réfutation par coupure.

   b) En déduire que pour chaque $c \in C_t$ et chaque $c' \in C_p$, ne construire qu'une seule des clauses déductibles à partir de c et c' (plutôt que toutes) est suffisant pour conserver la correction de derive_clause_vide.

   c) Modifier derive_clause_vide de façon à ignorer les clauses tautologiques.

Partie 2 *Le retour du club écossais*

L'objectif de cette partie est de faire prouver un résultat sémantique à votre machine à l'aide de derive_clause_vide (sous réserve que cette fonction soit correcte, résultat que l'on délègue à la partie 3).

Il existe en Ecosse un club très fermé dont les membres obéissent aux règles suivantes :

(1) Tout membre non écossais porte des chausettes rouges.

(2) Tout membre portant des chaussettes rouges porte un kilt.

(3) Les membres mariés ne sortent pas le dimanche.

(4) Un membre sort le dimanche si et seulement si il est écossais.

(5) Tout membre qui porte un kilt est écossais et est marié.

(6) Tout membre écossais porte un kilt.

10. Modéliser les règles du club en logique propositionnelle.

11. En déduire un ensemble de clauses S tel que S est satisfiable si et seulement si il est possible d'entrer dans le club. En utilisant la partie 1 et sans dresser de table de vérité à 32 lignes (!), monter que personne ne peut entrer dans le club.

Partie 3 *Correction et complétude de la réfutation par coupure*

L'objectif de cette partie est de montrer la correction et la complétude de la réfutation par coupure ainsi que la correction et la terminaison de la fonction derive_clause_vide.

12. On commence par montrer la correction de la réfutation par coupure.

    a) Montrer que pour toutes clauses C_1, C_2, C, si $\dfrac{C_1 \qquad C_2}{C}$ cut alors $\{C_1, C_2\} \models C$.

    b) Prouver que pour tout ensemble S de clauses et toute clause C, si $S \vdash C$ alors $S \models C$.

    c) En déduire en particulier que si $S \vdash \bot$ alors S est non satisfiable et conclure.

13. On prouve ensuite la complétude : si S est un ensemble de clauses non satisfiable alors il en existe une réfutation par coupure.

    a) Montrer que, si S est un ensemble de clauses non satisfiable ne contenant pas la clause vide, il existe $C_1, C_2 \in S$ et une variable v intervenant positivement dans C_1 et négativement dans C_2.

    Si w est une variable et S un ensemble de clauses, on note S_w le sous ensemble de S de toutes les clauses qui contiennent w. On appelle résolvant de S_w, noté $\mathrm{Res}(S_w)$ l'ensemble de toutes les clauses obtenues par coupure sur w à partir des clauses de S_w.

    b) Si S est un ensemble de clauses S et w une variable, montrer que S est satisfiable si et seulement si $\Sigma_w = (S \setminus S_w) \cup \mathrm{Res}(S_w)$ est satisfiable.

    c) En déduire que tout ensemble fini et non satisfiable de clauses est réfutable par coupure.

    d) On admet le théorème suivant :

    Théorème de compacité : Soit Γ un ensemble de formules du calcul propositionnel (potentiellement infini). Alors, Γ est satisfiable si et seulement si tout sous-ensemble fini de Γ l'est.

    En utilisant le théorème de compacité et les questions précédentes, montrer la complétude de la réfutation par coupure.

On vient de prouver que $S \vdash \bot$ si et seulement S est non satisfiable. Pour montrer que derive_clause_vide permet effectivement de décider si un ensemble de clauses est non satisfiable, il ne reste plus qu'à montrer que l'ensemble C_p calculé par cette fonction sur une entrée S contient la clause vide si et seulement si la clause vide est prouvable à partir de S.

14. Montrer la terminaison de la fonction derive_clause_vide.

15. Montrer l'invariant suivant : toute clause déductible par coupure (en une étape) à partir de l'ensemble de clauses C_p est soit dans C_t, soit dans C_p.

16. En déduire que, si derive_clause_vide est appliquée à l'ensemble de clauses S, en fin d'algorithme l'ensemble de clauses C_p vérifie : $C \in C_p$ si et seulement si C est prouvable à partir de S. Conclure.